Querying the SAINTETIQ summaries
Dealing with null answers[1]

W. A. Voglozin, G. Raschia, L. Ughetto and N. Mouaddib
LINA-INRIA-Polytech'Nantes-Université de Nantes / ATLAS-GRIM
2 rue de la Houssinière, BP 92208, 44322 Nantes Cedex 3, France
Email:{voglozin,raschia,ughetto,mouaddib}@univ-nantes.fr

***Abstract*— The data summarization techniques are usually not provided with tools for end-users to efficiently use the produced summaries. In a previous work, a querying tool for the SAINTETIQ summarization model has been outlined. It takes advantage of the hierarchical structure of those summaries to efficiently describe tuples that satisfy some selection criteria. This querying mechanism can be seen both as a *boolean* one over the summaries, and as a *flexible* one over the underlying relational tuples. This paper mainly investigates the case of null answers. It is shown how, in this case, alternative queries with non-null answers can be proposed to the user.**

I. INTRODUCTION

In order to handle the growth in size of databases, many approaches have been developed to extract knowledge from huge databases. One of these approaches consists in summarizing data (e.g., see [1], [2], [3], [4], [5]). However, summarization techniques are usually not provided with tools for end-users to efficiently use the summaries. As a consequence, users have to directly interpret the summaries, which is conceivable with a few summaries only. In other cases, tools are necessary.

In this paper, the structured data summarization model SAINTETIQ[2] developed in our research team [5] is considered. This model provides a compact representation of a database under the form of summaries organized in a hierarchy.

This paper proposes a querying mechanism for users to efficiently exploit the hierarchical summaries produced by SAINTETIQ. The first idea is to query the summaries using the vocabulary used in the summarization process and taking advantage of the hierarchical structure of the summaries. The querying process answers queries in which the criteria specify labels from the vocabulary. The algorithms perform boolean set comparisons and use the tree structure to cut branches and quickly reduce the search space. This leads to an important gain in response time, especially in case of a null answer (i.e., of an empty result set), as only a small part of the summaries hierarchy has to be explored, instead of the entire relational table.

Querying the summaries as explained above is interesting in order to rapidly get a rough idea of the properties of tuples in a relation. But sometimes, queries may have a null answer. Apart from the cases where null answers are unacceptable, the user has to think about another query, which might fail as well, and so on. Thus, the second idea is to find ways to provide an approximate answer when the user's query has an empty result set.

The next section presents an overview of the SAINTETIQ model, describing the representation of summaries and the main steps of the summary hierarchy building process. Section III thoroughly explains how advantage can be taken from the use of a SAINTETIQ summaries hierarchy in a flexible querying process. The expression of queries, the search procedure and the expression of results are briefly reviewed. Then, Section IV describes two possibilities for electing alternative queries for a null answer query.

II. THE SAINTETIQ SUMMARIES

The targeting of database records in flexible queries may lead to important response times when a large number of records is involved, or when subqueries are expressed. A long wait is frustrating, particularly when the query fails.

Database summaries offer a means of significantly reducing the volume of input for processes that require access to the database. The response time benefits from the downsizing. Furthermore, for this querying process, performance does not depend on specific combinations of attributes, i.e., whether the attributes are indexed or not, since the SAINTETIQ hierarchies of summaries are general indexes for the underlying data [6].

When querying the summaries, the gain in response time is made clearly at the expense of a loss of precision in the answer. This is of no importance when only a rough answer is required. When more details about the tuples are needed, querying the summaries can be considered a first step only: the entire set of relevant tuples can be easily retrieved from the answer summaries. Then, the querying mechanism remains efficient, and there is no loss of precision in the answer.

A. Running example

A single example illustrates the whole paper. It considers relation R = (thickness, hardness, temperature) from a `MATERIALS` table. A tuple from R describes a material used in an imaginary metallurgy plant to produce square sheets. Attribute `thickness` is expressed in mm and has a limited range (from 0.15 to 50). Attribute `hardness` is the final product's expected value on scale B of the Rockwell hardness test. Attribute `temperature` is a material's melting point.

[1]This research was partially supported by the French Ministry of Research and New Technolologies under the ACI program devoted to Data Masses (ACI-MD), project #MD-33.

[2]See URL: <http://www.simulation.fr/seq>.

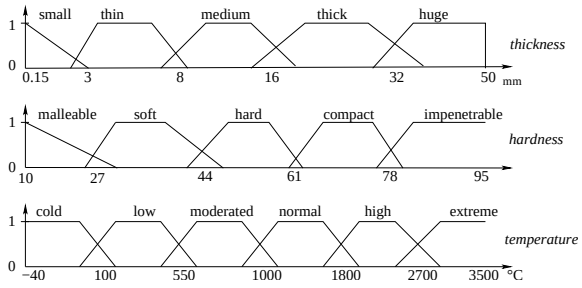

Fig. 1. *Linguistic variables for the* MATERIALS *table*

The linguistic variables associated with the attributes of R are shown on Fig. 1. Consider the MATERIALS table composed of the 5 tuples: $t_a = \langle 10, 38, 900 \rangle$ (copper and zinc alloy: CuZn40), $t_b = \langle 7.5, 40, 850 \rangle$ (copper and tin alloy: CuSn12), $t_c = \langle 12, 46, 896 \rangle$ (copper and arsenic alloy: CuAs05), $t_d = \langle 10, 35, 1530 \rangle$ (iron: Fe) and $t_e = \langle 5, 35, 1453 \rangle$ (nickel: Ni). Rewriting these 5 tuples using the linguistic variables leads to the candidate tuples shown on Fig. 2. The generated summaries that appear in the hierarchy on Fig. 3 are described on Fig. 4.

Material	Candidate tuples
UZ40	$t_{a_1} = \langle$ 0.7/medium, 1.0/soft, 0.85/moderated $\rangle$
CuSn12	$t_{b_1} = \langle$ 0.15/medium, 0.9/soft, 1.0/moderated $\rangle$ $t_{b_2} = \langle$ 0.45/thin, 0.9/soft, 1.0/moderated $\rangle$
CuAs05	$t_{c_1} = \langle$ 1.0/medium, 0.3/soft, 0.9/moderated $\rangle$ $t_{c_2} = \langle$ 1.0/medium, 0.5/hard, 0.9/moderated $\rangle$
Fe	$t_{d_1} = \langle$ 0.7/medium, 1.0/soft, 0.85/normal $\rangle$
Ni	$t_{e_1} = \langle$ 1.0/thin, 1.0/soft, 0.96/normal $\rangle$

Fig. 2. *Translation of tuples from table* MATERIALS

B. Summaries in SAINTETIQ

The SAINTETIQ model, founded on an incremental classification algorithm [5], aims at apprehending the information from a database in a synthetic manner. This is done through linguistic summaries structured in a hierarchy. The hierarchy building process consists in two steps.

First, the tuples are translated using the linguistic variables (which are part of a background knowledge provided by the user). Each attribute value of a tuple is rewritten using one of the corresponding linguistic labels. The rewritten tuples are called *candidate tuples*. As one attribute value may correspond to more than one fuzzy label (e.g., *8 mm* is described by *medium* and *thin*), one tuple (for instance t_b and t_c on Fig. 2) may yield many candidate tuples.

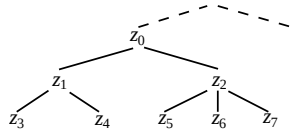

Fig. 3. *Part of the summary hierarchy for* MATERIALS

Second, each candidate tuple is incorporated into a tree and reaches a leaf node. This can be seen as a classification of the candidate tuple. It is important to notice that the tree is modified throughout candidate tuples incorporation: it progressively becomes a complete representation of the data. The way the tree evolves is partially controlled by learning operators (see [6]). These operators create and merge summaries so that the hierarchy reflects the current set of discovered concepts.

In the hierarchy structure, a level can be associated with the relative proportion of data that is described by a summary: the deeper the summary in the tree (or the lower its level in the hierarchy), the finer the granularity. Thus the lowest level contains the most precise and specific summaries. Such summaries are similar to candidate tuples in their intensional expression ($z = \langle \alpha_1/d_1, \alpha_2/d_2, \ldots, \alpha_n/d_n, \rangle$): there is only one label per attribute (for instance z_3 in Fig. 4).

By contrast, the root of the tree is the most general summary. It covers all data. The intensional expression of non-leaf summaries has one or more multi-valued attributes (e.g. z_1 and z_2). These labels are all the ones in the children summaries.

Summary	Intension
z_3	$\langle$ 1.0/medium, 1.0/soft, 1.0/moderated $\rangle$
z_4	$\langle$ 0.7/medium, 1.0/soft, 0.85/normal $\rangle$
z_5	$\langle$ 0.45/thin, 0.9/soft, 1.0/moderated $\rangle$
z_6	$\langle$ 1.0/medium, 0.5/hard, 0.9/moderated $\rangle$
z_7	$\langle$ 1.0/thin, 1.0/soft, 0.96/normal $\rangle$
z_1	1.0/medium, $\langle$ 1.0/soft, $\rangle$ 1.0/moderated + 0.85/normal
z_2	1.0/thin + 1.0/medium, $\langle$ 1.0/soft + 0.5/hard, $\rangle$ 1.0/moderated + 0.96/normal
z_0	1.0/thin + 1.0/medium + 0.7/thick, $\langle$ 1.0/soft + 0.5/hard, $\rangle$ 1.0/moderated + 0.96/normal + 0.75/high

Fig. 4. *Description of some summaries*

III. DESCRIPTION OF THE QUERYING PROCESS

The first step of a database flexible querying process that uses soft computing consists in extending criteria [7]. Thanks to linguistic variables defined for each attribute, criteria extension is already performed in SAINTETIQ. Then, binary operators can be used to identify the summaries (hence, the data) to be considered as results. This section deals with all aspects of selection, from the expression and meaning of a query to its matching against summaries.

A. Expression of a query

This approach to flexible querying intends to answer questions such as "what are thin materials like?" or "how are normal-temperature and soft-hardness materials?". In the prototype developed for querying, the questions are expressed using a user-friendly interface that composes the corresponding query in an SQL-like language. For the two previous questions, the queries are respectively:

```
Q1:  SELECT temperature, hardness
     FROM MATERIALS
     WHERE thickness IN ('thin')
Q2:  SELECT thickness
     FROM MATERIALS
     WHERE temperature IN ('normal')
     AND hardness IN ('soft')
```

Because an answer, for example "thin materials have a soft hardness and a normal temperature", is a description of basic data (summaries, candidate tuples, database records), description is considered as an elementary operation. Embedding the description operation (and others from summary-based querying) in an extension of SQL is a future project.

For a more formal expression of a query, let:

- S be a set of attributes;
- $\mathsf{R}(S)$ be the relation whose tuples are summarized;
- Q be a query, for instance Q_1 or Q_2;
- A_i be an attribute appearing in the query ($A_i \in S$);
- $d_{i,j}$ be a label (or descriptor) of attribute A_i, also appearing in the query.

A question explicitly defines some values (*thin*, *normal* or *soft*) called *required characters*. In a query, labels stand for required characters and serve as a basis for determining what data partake in the answer. A question also defines, sometimes implicitly, the attributes for which required characters exist. The set of these input attributes for a query is denoted by X. The expected answer is a description over the other attributes, whose set is denoted by Y. Without further precision, Y is the complement of X relative to S: $X \cup Y = S$ and $X \cap Y = \emptyset$.

Hence a query defines not only a set X of input attributes A_i but also, for each attribute A_i, the set C_i of its required characters. The set of sets C_i is denoted by C, as shown in the following example.

Example 1: Let Q_1 and Q_2 be the queries stated above. For each query, the sets are:

Q_1: $X = \{\text{thickness}\}$, $Y = \{\text{hardness, temperature}\}$, $C_{\texttt{thick}} = \{\text{thin}\}$ and $C = \{C_{\texttt{thick}}\}$

Q_2: $X = \{\text{hardness, temperature}\}$, $Y = \{\text{thickness}\}$, $C_{\texttt{hard}} = \{\text{soft}\}$, $C_{\texttt{temp}} = \{\text{normal}\}$ and $C = \{C_{\texttt{hard}}, C_{\texttt{temp}}\}$.

When users formulate a question, they expect data with some characteristics to be put forward. The meaning of that question remains an open problem. But for now, we consider a disjunctive semantics for criteria over the same attribute and a conjunctive semantics for criteria over different attributes.

B. Evaluation of a query

This section deals with matching one particular summary against a query to decide whether it corresponds to that query and can then be considered as a result. The query is transformed into a logical proposition P used to qualify the link between the summary and the query. P is in a conjunctive form in which all descriptors are literals. Then, each set of descriptors yields one corresponding clause.

Example 2: The query for "how are the materials which are thin or medium-thickness and moderated or normal-temperature?" is Q_5: SELECT hardness WHERE thickness IN ('medium', 'thin') AND temperature IN ('normal', 'moderated').

In Q_5, $X = \{thickness, temperature\}$, $C_{\texttt{thick}} = \{thin, medium\}$ and $C_{\texttt{temp}} = \{moderated, normal\}$. It follows that $P_5 = (thin \vee medium) \wedge (moderated \vee normal)$.

Let v be a valuation function. It is obvious that the valuation of P depends on the summary z: a literal d in P is positively valuated ($v(d) = \texttt{TRUE}$) if and only if d appears in z. Thus $v_z(P)$ denotes the valuation of P in the context of z.

Let $\mathcal{L}_{A_i}(z)$ be the set of descriptors that appear in z. Interpreting P relatively to query Q leads to discarding summaries that do not satisfy P. But, as shown in the following example, some summaries that satisfy P might not match the intended semantics of the query.

Example 3: Fig. 5 shows the characteristics of materials (t_{b_2}, t_{c_2}, t_{e_1}) covered by summary z_2 from table on Fig. 4. Suppose that z_2 is tested for conformance with a query Q_6: SELECT temperature WHERE thickness IN ('medium') AND hardness IN ('soft'). Then $P_6 = (\text{'}medium\text{'}) \wedge (\text{'}soft\text{'})$, and even though $v_{z_2}(P_6) = \texttt{TRUE}$, no material matches Q_6, as shown on Fig. 5.

Candidate	thickness	hardness
t_{b_2}	thin	soft
t_{c_2}	medium	hard
t_{e_1}	thin	soft
z_2	{thin, medium}	{soft, hard}

Fig. 5. *Example of descriptor combination*

The five relative situations of two sets being compared are displayed in Fig. 6. For convenience, they are grouped in three subfigures (a), (b) and (c). Confronting a summary z with a query Q involves comparing $\mathcal{L}_{A_i}(z)$ and C_i for each attribute from Q. Only three cases may occur:

- **Case 1:** at least one attribute falls in (a). z does not match the semantics of the query: $v_z(P) = \texttt{FALSE}$.
- **Case 2:** all comparisons fall in (b). z matches the semantics of Q. The following expression holds: $v_z(P) = \texttt{TRUE} \wedge \forall i,\ \mathcal{L}_{A_i}(z) \subseteq C_i$. z is considered as a result if all attributes are one-valued (the summary is a leaf). Otherwise, each leaf-summary in the sub-tree of z is a result. This condition helps avoid the mistake emphasized in example 3.
- **Case 3:** at least one comparison falls in (c) and the others fall in (b). The presence of required characters in each attribute of z suggests, but does not guarantee, that results may be found in the subtree starting from z. Exploration of the subtree is necessary to retrieve possible results; each branch will end up in either case 1 or case 2.

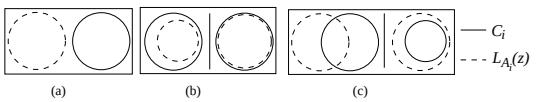

Fig. 6. *Comparison of descriptor sets $\mathcal{L}_{A_i}(z)$ and C_i*

C. Selection algorithm

This section presents the algorithm that applies the matching procedure from the previous section for a specific query. The selection (algorithm 1), based on a depth-first search, is complete thanks to a property of the hierarchy: the generalization step in the SAINTETIQ model guarantees that any descriptor that exists in a node of the tree also exists in each parent node. By contrast, a descriptor is absent from a summary's intension if and only if it is absent from all subnodes of this summary. This property of the hierarchy (which can be easily seen on Fig. 4) permits branch cutting as soon as it is known that no result will be found. In all cases, all relevant results, and only relevant results, are captured.

Algorithm 1 describes the exploration and selection function with the following assumptions:

- function *Explore-Select* returns a list of summaries;
- function *Corr* symbolizes the matching test reported in Section III-B;
- operator '+' performs a list concatenation;
- function *Add* adds an element to a list;
- L_{res} is a local variable.

Algorithm 1 Function Explore-Select(z, Q)

$L_{res} \leftarrow \langle\rangle$ {the list for this subtree is empty}
if Corr$(z, Q) =$ *indecisive* **then**
  for all child node z_{child} of z **do**
    $L_{res} \leftarrow L_{res}+$ Explore-Select(z_{child}, Q)
  end for
else
  if Corr$(z, Q) =$ *exact* **then**
    if z is a leaf node **then**
      Add(z, L_{res})
    else
      $L_{res} \leftarrow L_{res}+$ Explore-Select(z_{child}, Q)
    end if
  end if
end if
return L_{res}

Example 4: The result of applying the algorithm on the portion of hierarchy in Fig. 3 for some queries is listed below:

Query	Result list	Tuples
Q_1	$\langle z_5, z_7\rangle$	t_{b_2}, t_{e_1}
Q_2	$\langle z_4, z_7\rangle$	t_{d_1}, t_{e_1}
Q_3	$\langle z_3, z_4, z_5, z_7\rangle$	$t_{a_1}, t_{b_1}, t_{c_1}, t_{d_1}, t_{b_2}, t_{e_1}$
Q_4	$\langle\rangle$	–
Q_5	$\langle z_3, z_4, z_5, z_6, z_7\rangle$	$t_{a_1}, t_{b_1}, t_{c_1}, t_{d_1}, t_{b_2}, t_{c_2}, t_{e_1}$
Q_6	$\langle z_5, z_7\rangle$	t_{b_2}, t_{e_1}

D. Classification: presenting the results

The classification step is an aggregation of selected summaries according to their interpretation with respect to proposition P: summaries that have the same required characters on all attributes of the input attributes set X constitute a class. A class is equivalent to a group in an SQL SELECT ... GROUP BY... statement except the aggregation operator is a union of labels instead of the usual operators (COUNT, SUM, MIN, etc.).

Example 5: Consider query Q_5 from example 2. Proposition P_5 induced by Q_5 (see Example 2) admits 9 different interpretations (i.e. sets of variables valuated to `TRUE` so that the proposition is satisfied) although only four interpretations are shown in the table below. The other interpretations are {thin, medium, normal}, {thin, medium, moderated}, {thin, moderated, normal}, {thick, moderated, normal} and {thin, medium, moderated, normal}.

However, grouping several tuples from a relation R(S) on a subset X of S causes different values to appear in each group for attributes in $S - X$, due to the unicity of each summary. When such values exist for a query, they are grouped in a class as shown in the following table for the {medium, moderated} class.

Interpretation	Summaries	Result
{thin, moderated}	z_5	soft
{thin, normal}	z_7	soft
{medium, moderated}	z_3, z_6	soft, hard
{medium, normal}	z_4	soft

Aggregation of summaries inside a class (for instance, {medium, moderated}) is a union of descriptors: for each attribute A_i of output set Y, the querying process supplies a set of descriptors. This set characterizes summaries that respond to the query through the same logical interpretation (i.e., summaries that show the same labels for input attributes).

As a response to a query, the process returns a list of classes along with a characterization of the class for each output attribute. The list is interpreted as follows: *while searching for thin or medium-thickness and normal or moderated-temperature materials, it turned out that:*

- *thin moderated-temperature materials are soft;*
- *medium-thickness moderated-temperature materials are either soft or hard;*
- ...

The use of classes has a few benefits: i) the results are expressed in an intensional way ii) one can easily identify which class accounts for an output label; and iii) it remains possible to provide a unique list of output labels by performing a union of labels from all classes.

IV. REPAIRING QUERIES

The intention, in an attempt to repair queries, is to offer an answer even when no summary corresponds to the query, in the strict sense considered so far. Repairing queries has already been implemented in the context of a mediator by Bidault et al. [8], [9], as one of the cooperative aspects reviewed by Gaasterland et al. in [10].

Our view of repair consists in a modification of the original query. This modification is performed from the optimistic idea that there exist results semantically close to those targeted by the user. In order to select such approximate-result summaries, the query is modified using the parsed SAINTETIQ summaries or pre-established information.

A. Query modification

This procedure happens each time a tree exploration that is conducted to find answers for a query Q fails at a specific

node z (i.e., when some required characters are absent from z). z is then called a failure node. As the exploration may fail for more than one summary, several modifications of the same query can be proposed to the user.

The first strategy consists in finding substitutes for the *missing* characters within a limit imposed by a distance described in Section IV-B. A substitution query denoted by Q^*, is derived from the original query Q. However, no guarantee can be given as to the existence of results for the new query. An intuitive possibility for substitutions lies in the linguistic variables: absent descriptors are replaced by the closest ones in a failure node. However, it requires an order over the considered attribute domain. In case no intuitive order exists, another possibility consists in defining a similarity relationship matrix over the attribute domain. In this case, substitutions may have an associated weight that would allow a finer distinction between results from several alternative queries.

In both cases, if failure at summary z leads to a label d being replaced by a label d^*, d will not be used later as a substitute of d^* at another failure point below z (i.e., in the subtree from z). Indeed, if z is a failure node, a property of the summary hierarchy guarantees that no summary in the subtree from z can be a result to the query.

The second strategy is guided by the summary hierarchy and it guarantees that results will be found for the new query. It is more flexible and more adapted to an interactive mode.

B. Distance between queries

In order to avoid a series of query modifications that would lead to irrelevant alternative queries, a measure of distance between queries is introduced. Only the *closest* alternative queries to the initial one are proposed to the user. Assume that a query Q can be associated to a bit string in which each bit reflects the presence or the absence of a label in Q.

Definition: Let Q and Q^* be two queries respectively associated to bit strings S and S^*. The distance between Q and Q^*, denoted by $d(Q, Q^*)$, is the number of 1s in S XOR S^*. That number accounts for the modifications (insertions or deletions of labels) that are necessary to obtain Q^* from Q. This Hamming distance [11] satisfies the following properties:

1) $d(Q, Q) = 0$
2) $Q \neq Q^* \Rightarrow d(Q, Q^*) > 0$
3) $d(Q, Q^*) = d(Q^*, Q)$
4) $d(Q, Q^*) < \sum_{A_i \in C} |D_{A_i}|$ where D_{A_i} is the set of terms for the linguistic variable over attribute A_i.

However, this distance is not fine-grained enough to allow an automatic process. The user may have to guide the search because the distance does not take into account the relative closeness between two labels. For instance, while looking for materials with a *low* temperature, modified queries that target *cold* or *high* temperature materials are considered equivalent.

C. Query substitution algorithm

This section presents the query modification procedure described by algorithm 2. The procedure is a variant of algorithm 1 (see Section III-C) cause it adds an additional processing to case 1 (from Section III-B). This processing is the *Modify* function. Usage of available information (such as linguistic variables or a similarity matrix) is made in that function.

In this algorithm, Q_{ref} represents the original query expressed by the user. It is used in function to guarantee that the fourth property in Section IV-B will still hold after the modification. Q is the current query being evaluated, initially equivalent to Q_{ref}. *exact* stands for case 2 (from Section III-B) and *indecisive* stands for case 3.

Algorithm 2 Function Explore-Select-Modify(z, Q, Q_{ref})

$L_{res} \leftarrow \langle\rangle$
if Corr$(z, Q) =$ *indecisive* **then**
  for all child node z_{child} of z **do**
    $L_{res} \leftarrow L_{res}+$ Explore-Select-Modify(z_{child}, Q)
  end for
else
  if Corr$(z, Q) =$ *exact* **then**
    Add(z, L_{res})
  else {no correspondence, but the summary might be acceptable}
    if Can-Be-Modified(Q, Q_{ref}) = `TRUE` **then**
      Q^* = Modify(Q, z)
      $L_{res} \leftarrow L_{res}+$ Explore-Select-Modify(z, Q^*, Q_{ref})
    end if
  end if
end if
return L_{res}

Example 6: Consider for instance query Q_7*: SELECT temperature WHERE thickness IN ('medium') AND hardness IN ('compact')*. This query fails as $v_{z_0}(P_7) = FALSE$ and the failure node z_0 is added to the list. Then, from z_0, two alternative queries can be proposed : Q'_7*: SELECT temperature WHERE thickness IN ('medium') AND hardness IN ('hard')* and Q''_7*: SELECT temperature WHERE thickness IN ('medium') AND hardness IN ('soft')*, as the two labels 'hard' and 'soft' appear in z_0. Both alternative queries have distance 1 from Q_7, but the first one is closer w.r.t. the similarity on hardness.

D. Summary-guided modification

Whenever a query fails, its evaluation allows to detect the *reasons* of the failure. A reason stands for an attribute, from the query, whose criterion has not been fulfilled. The detection is immediate from the set-based comparison detailed in Section III-B and illustrated by Fig. 6.

During a search to answer a query Q_0, the exploration of a summary hierarchy may fail for each child node of a summary z_1. However, z_1 can still be considered as the best approximation of a result to Q_0 on the branch that led to z_1. Since a hierarchy exploration is made on a step-by-step decision making process and z_1 is the last point where a *next valid path* had to be found, it is reasonable to assume that a query Q_1, which has z_1 as a result, is close to Q_0 provided both queries require the same attributes. Thus, a failure node in the summary tree yields one new alternative query.

Queries considered close to Q and derived from failure points offer a guarantee of results, which was not the case for Q_0. Nevertheless, using algorithm 1 to implement that proximity assumption requires two steps: i) determining that

the query Q_0 has no results and ii) searching for the failure points in order to derive the modified queries. The new algorithm 3 performs the two steps by assuming from the beginning that the query being evaluated will have no result. While exploring the tree, two lists are built, one that contains the results to Q_0 (L_{res}) and another one that contains failure nodes (L_Q). The latter is useful when the first is empty only. Building the list of failure nodes is made at a low cost. It is not penalizing when the query has answers, and avoids a second parsing of the summaries in the other case.

E. Expression of results

Generally, the algorithm produces several alternative queries, more or less close to the original one. These queries can be ordered according to their closeness to the original one, measured either by the distance in section IV-B or taking into account the closeness of the modified labels.

The query modification procedure appears as a relaxation of search criteria, for the new query is more general. However, each substitution of a query Q by a query Q^* is local as shown by example 7.

The *local* feature of modification is justified first by the fact that a failure occurs at a specific node in the search tree and second, by the relatively high number of failures that naturally occur in a efficient branch-cutting search procedure. Applying all substitution queries over the whole tree would be not only expensive but also unjustified for the part of the search tree that is not concerned with a failure.

Algorithm 3 Function Guided-Sel(z, myCorr, Q)

```
L_res ← ⟨⟩
if myCorr = exact then
   Add(z, L_res)
else
   if myCorr = indecisive then
      for all child node z_child of z do
         Corr_f = Corr(z_child, Q)
         L_res ← L_res+ Guided-Sel(z_child, Corr_f, Q)
      end for
      if L_res = ⟨⟩ then
         Add(z_child, L_Q)
      end if
   end if
end if
return L_res
```

Example 7: Consider a hierarchy with root z_0 searched in order to answer a query Q_0 and two failure nodes z_1 and z_2 are found. As described above, two new queries Q_1^* and Q_2^* are induced.

The location of the selected summaries can be categorized depending on the query (Q_0, Q_1^*,Q_2^*) they are a result of: some are in z_0's subtree, others are in z_1's and others are in $z_2's$.

Selected summaries that answer Q_1^* belong to the subtree starting from z_1 even though there might be answers to Q_1^* in the rest of the hierarchy. The same holds for Q_2^*.

For short, algorithm 3 follows these steps:

1) evaluation of the initial query Q;
2) no result, execution of the modification;
3) determination of the best substitution query Q^*, either using the distance measure or interactively;
4) evaluation of Q^*;
5) expression of results of Q^*.

V. Conclusion and future research

In this paper, a querying tool for the summarization model SaintEtiQ has been proposed. It allows end-users to efficiently retrieve summaries, and exploits the hierarchical structure of the summaries produced by SaintEtiQ.

From an algorithmic point of view, the querying process explores a summary hierarchy. It performs a set comparison between the summary and the query on the basis of linguistic labels from a user-defined vocabulary. The result of the comparison determines whether the summary is a result but also whether a part of the hierarchy will be explored. The search procedure is therefore a classical boolean tree exploration with branch-cutting algorithm whose novelty lies in the use of summaries. The querying machinery, as well as a user-friendly interface have been developed.

An extension to this method is also proposed. It allows to answer queries that have empty result sets with semantically close data. Two strategies have been considered to reach that goal: exploiting some available information, or using the summaries to determine alternatives to the original query.

This work is a step towards a more complete flexible query answering system. Clearly, the richness of the framework is far from being entirely exploited yet. Several future developments, such as ranking the results, introducing preferences or priorities, or the possibility to use another vocabulary, are under consideration. Expressiveness remains a main point for future work. Indeed, it would be of great interest to allow some imprecision in the user queries, and not just in the representation of information.

References

[1] J. C. Cubero, J. M. Medina, O. Pons, and M. A. V. Miranda, "Data summarization in relational databases through fuzzy dependencies," *Information Sciences*, vol. **121 (3-4)**, pp. 233–270, 1999.

[2] D. Dubois and H. Prade, "Fuzzy sets in data summaries — outline of a new approach," in *Proc. of IPMU'2000*, 2000, pp. 1035–1040.

[3] J. Kacprzyk, "Fuzzy logic for linguistic summarization of databases," in *Proc. of FUZZ-IEEE'99*, 1999, pp. 813–818.

[4] D. H. Lee and M. H. Kim, "Database summarization using fuzzy ISA hierarchies," *IEEE Trans. on Systems, Man and Cybernetics-Part B: Cybernetics*, vol. **27**, pp. 68–78, Feb. 1997.

[5] G. Raschia and N. Mouaddib, "SaintEtiQ: a fuzzy set-based approach to database summarization," *Fuzzy Sets And Systems*, vol. **129**, pp. 137–162, 2002.

[6] G. Raschia, "SaintEtiQ: une approche floue pour la génération de résumés à partir de bases de données relationnelles," Thèse de doctorat, Université de Nantes, Dec. 2001.

[7] H. L. Larsen, "An approach to flexible information access systems using soft computing," in *Proc. of the 32nd Hawaii Int. Conf. on System Sciences*, vol. 6, Jan. 1999.

[8] A. Bidault, C. Froidevaux, and B. Safar, "Repairing queries in a mediator approach," in *Proc. of ECAI'00*, 2000, pp. 406–410.

[9] ——, "Similarity between queries in a mediator," in *Proc. of ECAI'02*, 2002, pp. 235–239.

[10] T. Gaasterland, P. Godfrey, and J. Minker, "An overview of cooperative answering," *J. of Intelligent Systems*, vol. **1**, no. 2, pp. 123–157, 1992.

[11] R. W. Hamming, "Error detecting and error correcting codes," *The Bell System Technical Journal*, vol. **27**, no. 2, pp. 147–160, Apr. 1950.